

British Amateur Television Club



PIC Programming:

Home

Up

PIC Programming

A short introduction to the Microchip PIC series of microcontrollers by Brian Kelly, GW6BWX.

- ✚ [What is a microcontroller?](#)
- ✚ [Where do the PIC devices fit in the grand scheme of things.](#)
- ✚ [Who invented them?](#)
- ✚ [Variations on a theme.](#)
- ✚ [Why they were chosen as the basis for BATC projects.](#)
- ✚ [Internal PIC architecture.](#)
- ✚ [The instruction set.](#)
- ✚ [Software features and planning.](#)
- ✚ [Programming tips and tricks.](#)
- ✚ [From conception to publication.](#)

Prepared for the CAT'99 event on August 8th, 1999.

Many thanks to Microchip Technology Ltd for their co-operation while producing these notes and their permission to reproduce diagrams from their CD data book.

What is a microcontroller?



All the PIC families of devices, with the possible exclusion of the 17C42, fall into the category of electronic devices called microcontrollers. A microcontroller is a component that has at its core the same building blocks as a microprocessor but is optimised to interact with the outside world through on-board interfaces. A microprocessor is normally optimised to co-ordinate the flow of information between separate memory and peripheral devices which are located outside itself. Connections to a microprocessor include address, control and data busses that allow it to select one of its peripherals and send to or retrieve data from it. Because a microcontrollers processor and peripherals are built on the same silicon, the devices are self-contained and rarely have any bus structures extending outside their packages.

The 17C42 is unique in that it can work as a microcontroller or as a microprocessor or as a combination of both. This unusual arrangement is achieved by sacrificing some of the chips input/output (IO) pins, using them as extensions of its core processors internal busses.

Where do the PIC devices fit in the grand scheme of things.



Microchip have cultivated their own market by producing devices so versatile that they can replace almost all circuits that previously required several discrete devices. Their low pricing scheme made it feasible for designers to dispense with many combinatorial or sequential logic circuits, replacing them with a single circuit emulating the desired function by running a simple program. The simple programming language and free support tools have ensured that development time and expenses can match or undercut the cost of designing conventional circuits.

Although PIC microcontrollers are convenient and simple to use, their place in life must be understood to gain most benefit from them. They are not, and never will be efficient at complex mathematical functions for example, and are for practical purposes not able to handle huge amounts of data. These are tasks better suited to a conventional microprocessor, which can offload its storage demands and data manipulation load onto dedicated peripherals. On the other hand, using a PIC to emulate a simple logic gate would be inefficient and wasteful. Somewhere between these extremes there is a region where computing demand is more modest and process controlling is more important than number crunching, this is where PIC devices excel.



Who invented them?

This is probably open to dispute but the earliest devices I have been able to find are the PIC1650 and PIC1655 described in a Plessey handbook dating back to 1986. Although these devices are not listed by Microchip in their literature, their pin names, internal architecture and program mnemonics are almost identical to the present PIC16C5x devices. Plessey describe NMOS devices whereas the current production devices are all CMOS (that's what the "C" in the part name indicates) but Microchip claims to have shipped over 80 million NMOS chips prior to the CMOS fabrication taking over. Whether one company made them under license from the other or the product range migrated from one to the other is unknown.



Variations on a theme.

There are approximately one hundred different types of PIC microcontroller, each having its own special features. Broadly, they can be categorised into four families:

PIC12Cxx	8-bit program word
PIC16C5xx	12-bit program word
PIC16C6xx	14-bit program word
PIC17Cxx	16-bit program word

The xx's are the numbers of specific types within the family. Note that despite the different program word lengths, all devices have 8-bit data lengths. There is more about this under the "architecture" heading. The principle difference between members within a family are in the extra interfaces built into the silicon. Some devices have extra digital input/output pins (and hence larger packages) for examples while others have analogue to digital converters or counters on-board. Most of the devices are available in both conventional Dual-In-Line packages and in surface mount outlines although the larger pin count devices cannot be housed in DIL for mechanical reasons. To aid program development, the devices that are OTP (One Time Programmable) are also available in windowed ceramic housings so they can be erased and reused, these are the "JW" versions of the chips. Windowed devices are much more expensive and are intended to be used in development situations only, the finalised design going into a non-windowed device.



Why they were chosen as the basis for BATC projects.

The majority of video related projects use some logic circuitry, whether as part of a control system or intelligently operating a display of some sort. The low cost and wide availability of these devices made them eminently suitable for inclusion in new projects. The decision to adopt the PIC family was not taken collectively or singly by committee members but happened gradually as the devices started appearing increasingly in CQ-TV articles. With PIC expertise growing within the membership and opportunities for designing clever but simple projects with these chips, it seemed silly not to centre attention on them. Farnell and Maplin, probably the biggest two suppliers of components to the hobby market, carry stocks of most PIC variants with rapid delivery times making them very accessible. The similarity of the core instruction sets of all PIC devices also ensures an easy upgrade path to newer devices as older type become obsolete, ensuring longevity of new projects.



Internal PIC architecture.

PIC devices differ from most other microcontrollers in two ways, the layout of their internal circuit blocks and their instruction sets. Somewhat, these are related and both must be understood before their strengths can be utilised to best effect.

There are two “normal” architectures used in modern processors, the more common one is that used in the Z80 right through to the Pentium 3 chips, this is called the “Von Neuman” architecture. In this system there are three distinct signal pathways, also called busses. The data bus is used to carry parallel bits of data and instructions between the processor and any of its peripheral circuits. The address bus is used to identify which peripheral or storage location is the desired source or destination for the data. The control bus carries a variety of signals that set the direction of the data transfer, processor to peripheral or vice versa, and other connections which cause interrupts or transfer synchronisation. In this system, the program being executed and any data being used are stored in devices mapped into the same address space. They may of course be in entirely different types of memory. For example, the program may be in EPROM with the data in RAM but from the processors point of view they are reached the same way but at different addresses.

PIC devices use the second type of layout called the “Harvard” architecture. This differs from the first type in that the communication pathways for program bytes and data bytes are separate from each other. Rather than the instructions being executed and variables being stored sharing the same data and address busses, they have their own private pathways. The advantage of splitting the program instructions from the remainder of storage is that the chip can load instructions and data in parallel instead of serially in sequential memory fetches. As well as reducing the time taken to retrieve the instructions there is an additional benefit, that the instruction pipeline (queuing instructions so they are ready for immediate use) can be more efficiently implemented. Because the data and instructions follow different busses, there is no need for them to be the same bit-width. Internal registers and data transfers are eight bits wide although not all bits are available at the chips pins on some of the IO ports.

The remaining sections of the devices are similar in operation to conventional peripherals but of course occupy the same silicon slice as the core processor. The chips all incorporate an RTCC (Real Time Clock Counter) register that can be configured to count internal timing intervals or outside events. In “internal” mode it can be used to derive precise timing intervals or delays. Optionally a prescaler can be switched between the count source and the RTCC allowing a much wider range of delays or higher event count to be reached.

As a safety precaution, a watchdog timer is provided. Should this timer ever reach maximum count it will internally operate the chips reset system and cause it to restart. The feature is programmable; it can be enabled or disabled during the programming stage. If enabled, the program being executed would be expected to reset the counter before time-out occurred so the chip would never falsely restart.

The remaining internal registers are different from one PIC to another and the appropriate data sheet should be read to learn of specific details.

The instruction set.



PIC chips share an almost identical instruction set which makes program development and porting from one type to another a simple task. The instructions are mostly “orthogonal”, meaning that almost any operation can be performed in the same fashion to any part of the chip. For example, the same command to send data to one of the port pins can be used to send data to the RTCC, only the destination is different.

Although there are no strict rules about instruction set categories, broadly they fall into two types, CISC and RISC. CISC stands for “Complex Instruction Set Computer” and is exemplified by Intel’s X86 and Pentium processors which have close on one thousand instructions available. RISC stands for “Reduced Instruction Set Computer” such as the PIC range with as few as 33 instructions in their repertoire.

There are pros and cons to both types, arguably a CISC instruction does more in a single instruction because each instruction completes a complex task while RISC instructions are simpler and therefore more are needed, but each is faster to execute.

PIC devices all use instructions at the RISC end of the scale.

Although there are fewer instructions, their wider bit-width allows more to be packed into each one. All PIC instructions occupy one single address of storage and almost all can execute in one machine cycle (four clock cycles) making them extremely fast. It is worth comparing this to the 8086 with its slowest instruction needing 206 clocks and the Z80 needing 23. Similarly, the longest instruction on an 8086 occupies 6 addresses and the Z80 a mere 4. Don’t forget that each address of these instructions in Von Neuman architecture has to be accessed in sequence along a single data bus while the PIC Harvard architecture delivers all the instruction in one gulp.

The PIC instructions comprise several bit fields. The 12 or more bits in the instruction can be split into groups, defining the type of operation required, source of data and where to place the result. This can be clearly seen in the 16C84 instruction set listing that accompanies these notes. Having all this information present in one instruction provides the processor with all the details it needs to complete the operation, hence the rapid execution time.

Most instructions utilise the “W” (Working) register; this is equivalent to the accumulator in microprocessor terms. Whenever any of the byte-orientated commands are executed, one of the bits in the instruction field determines whether the result of the operation is placed in the W register or back in the source. This is very useful, in most other instruction sets the destination is fixed and requires further instructions to move results to their final place. Note that I am calling the holding places in the chip “registers” while Microchip call them “files”, use the two words interchangeably, even the manufacturers muddle them in their documentation.

There are a few caveats to mention, in the 16C5x instruction set there exists a “TRIS” command that sets the IO ports to normal or tri-state (open circuit) mode. The same instruction exists in the 16C8x devices but is not guaranteed to be included in later types, directly write or read the ports own control register instead to maintain forward compatibility.

Software features and planning.



The PIC instruction set will seem unusual to someone used to programming conventional microprocessors. It does take some time to come to terms with the odd mnemonics and assembler syntax but with patience, it starts to make complete sense. A complete novice would probably find it easier to grasp than someone with preconceptions would.

Writing programs is more to do with understanding the objective than understanding assembly language. The actual coding of a program is the simple part, understanding the route the program must take is what confuses and frustrates most people. A rule of thumb is spend 70% of the time planning the program, 10% writing it and 20% verifying and debugging it. Failure to complete the first stage will invariably cause grief later. It is a good idea to learn how to write flow-charts. A properly planned flow chart can be turned step for step into PIC instructions, by visualising the actions taken as the program executes you virtually decide which instructions to use. Each “landing place” where a flow arrow joins a box or line becomes a label (place

marker) in the program and each box becomes one or more instructions.

Depending on the nature of project, it is also well worth investigating which tasks are best suited to software solutions and which are better completed in hardware. For example if a signal needs to be upside-down, it may be a simple fix to invert it in software rather than add circuitry. On the other hand a high-speed counter may be beyond the capability of a PIC and better implemented in hardware. A good designer knows how to combine the strengths of hardware and software to best advantage.

As mentioned earlier, the PIC has instructions that work on any valid source and destination. The instruction to write to the RTCC is "MOVWF RTCC" and the instruction to write to one of the IO ports is "MOVWF PORTA", I've assumed these destinations are properly defined in the program but the instruction "MOVWF" is common to both. In fact, the same instruction works on all the possible destinations making it much easier to remember what it does. The same rule applies to all the other commands, if it works on one destination it will work on them all. The mnemonics may seem strange at first but in time you start to think of them as short sentences, "MOVWF" becomes "move W to file" which makes sense when taken in context of the remainder of the instruction.

If Microchips assembler is used, a rather nice feature is available, you can create your own mnemonics for single or groups of instructions. For example for page switching (a technique used to select different sets of registers) I usually create my own mnemonics called "page_0", "page_1" and so on which are copies of the BCF and BSF instructions applied to the option register. I find the page word far more meaningful than "BSF STATUS,RP1" or whatever. Once created they can be used like an official instruction anywhere in the program. Some programmers prefer to redefine the conditional jump instructions from "BTFSS" (bit test in file and skip if set) to "jump-if-zero" or some other words more meaningful to them. The assembler lets you redefine just about everything, including the standard set of instruction words.

Programming tips and tricks.



With care, it is possible to write very compact and efficient program code. There are a few tricks and short cuts worth remembering; here are a couple of examples:

9.1 Look-up tables.



When a number is in a list and needs to be accessed by its position from the start, use the RETLW instruction. It means "Return from subroutine with a literal in W". The trick is to put the "ADDWF,PCL" instruction at the beginning of the list then put the offset into the table in W and "CALL" the list. The orthogonal instruction set lets you perform maths on the program counter so the value in W advances the program that many instructions then returns with W loaded with the content of the list. For example to find the third entry in the table:

```
Look_up_table      ;this is label, the target for the call.
    ADDWF PCL,1      ;add W to program counter
    RETLW 12         ;retlw with data in the list
    RETLW 34
    RETLW 45
    RETLW 56
```

Then if the W register is loaded with 3 then "CALL Look_up_table" is used, the program stores the return address then jumps to the ADDWF instruction at the start of the table. The instruction adds the 3 to the program counter (PCL) so the next instruction it executes is "RETLW 45" which returns to the original program with 45 in W.



Shift registers.

Many projects either use or generate streams of serial information. This could be to communicate with other devices, for example an I2C bus or could be to interface to a computer serial port. A simple way to convert the contents of a register into serial bits is to rotate the register left or right with the “RLF” or “RRF” instructions which puts the first or last bit into the “C” (Carry) bit of the status register. The same or opposite instruction can then be used to shift it into one of the end bits of an IO port. Putting the shift inside a loop allows more than one bit to be shifted out.

```
Shift_out          ;loop back here if more bits to shift
RRF my_data,1      ;shift the byte to the right putting b0 in C
RLF PORTA,1        ;shift C into the IO port bit 0
```

Shifting in is best performed by using the bit testing instructions directly on the port bit being driven. Be aware that there is a side effect to directly shifting any dedicated register, all the other bits are shifted too! In some circumstances, it will be better to make a decision based on the state of the C bit then using the bit set and clear instructions to change the port bits.



From conception to publication.

Rarely at a rally or convention do I escape the “How do *you* do it?” question. The answer has changed over time as my programming techniques and budgets have changed. The same underlying course of events takes place though. Necessity is the mother of invention and necessity is what usually starts the design process rolling. I take as an example the video overlay generator in CQ-TV 187 which was needed as part of a much larger project where it would be used to provide station identification. The article in the magazine is derived from the original design both for my own pleasure and as it looked useful, to fill a few pages. When first produced it only had one message but also included software implemented keyed tone generators. The stages of development are typically:

1. Inspiration.
2. Head scratching for a few days, mulling over the different ways of tackling the project.
3. Drawing a flow chart or draft program as a list of tasks rather than program code.
4. Mentally debugging stage 3, looking for problems before they occur.
5. Typing the code, using chunks from earlier projects when appropriate.
6. Assembling the code with MPASM, eliminating typing and syntax errors.
7. Simulating the program flow with MPSIM to find logical errors.
8. Programming a chip, preferably one with EE memory or a windowed device.
9. Back to step six until it works properly.
10. Designing a PCB if one is needed.
11. Building, testing and repeating the debugging process if it has problems.
12. Drawing a double size schematic and digitising it.
13. Photographing the assembled PCB.
14. Typing the text of the article, inserting schematic, photo and other diagrams.
15. E-mailing the completed article to the Editor.

The whole process usually takes about three weeks, from start to finish although the last steps sometime have to be repeated until the article is ready for sending to the printers.

Of course, many of my projects are not likely to be of use to others so are never published. I doubt many people would want to build a new controller for a 30 year-old central heating boiler like mine for example!

I use a commercially available programming unit to burn the code into the PIC chips, unlike most experimenters, I also have to work with PAL and gate array devices so something a little more sophisticated than the BATC EPROM and PIC programmers is called for.

I hope these notes and the talk itself provide the inspiration for others to try their hand at PIC programming. It isn't difficult and can be very rewarding. Don't hesitate to contact me if you need help with PIC devices in general but please bear in mind that I cannot answer questions about programs except ones I have written myself.

[Click here](#) to view or download the PIC 16C84 data sheet. (pdf format)

[[Home](#)] [[Up](#)]

Last modified: 1999-08-10.

Please send any comments or suggestions to the



[BATC Webmaster.](#)



Copyright © 1997-99 by The British Amateur Television Club, unless otherwise stated.

[Terms of Use](#)